**ASSIGNMENT-2**

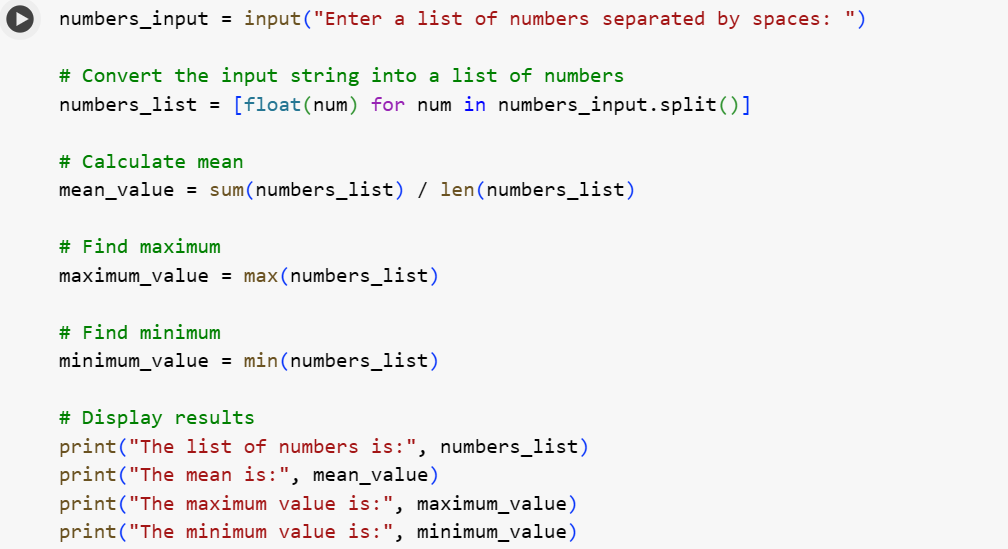
**NAME : G.SAI RAJ**

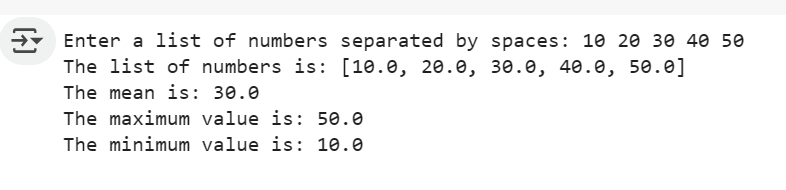
**HT NO: 2403A52149**

**BATCH:** 24BTCAIAIB06

**TASK-1:** Write a python program that reads a list of numbers and calculates mean,maximum and minimum values.

**CODE AND OUTPUT:**

****

****

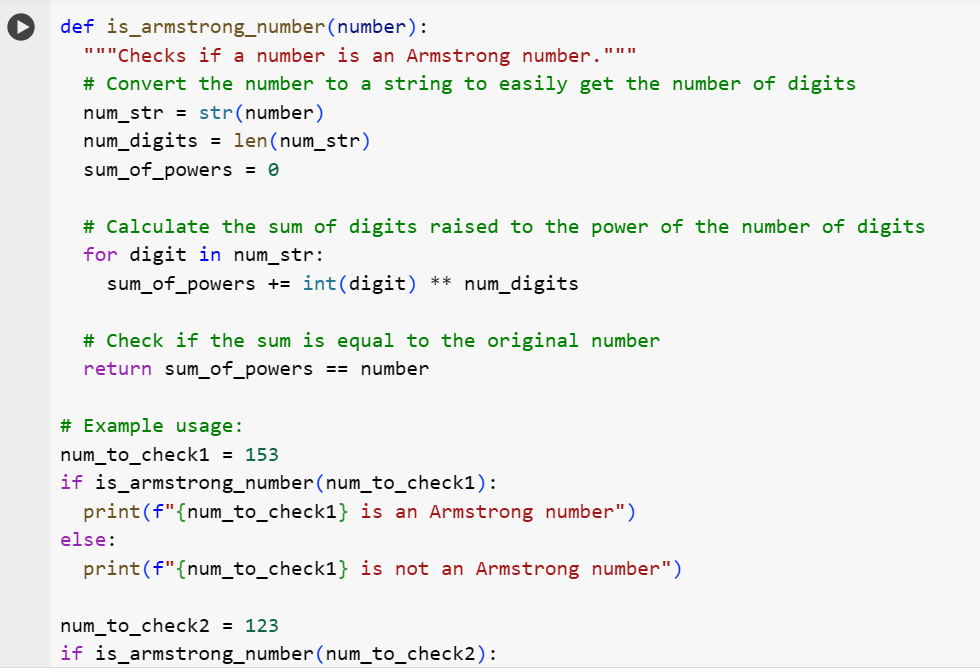
**EXPLANATION:**

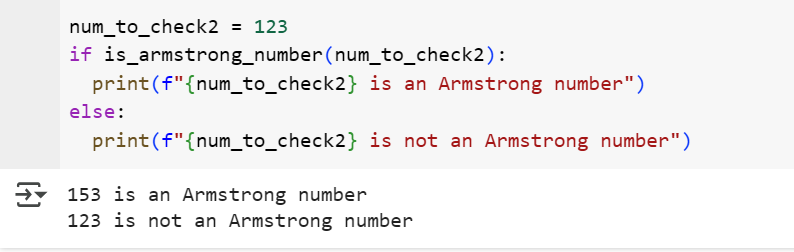
1. **numbers\_input = input("Enter a list of numbers separated by spaces: ")**: This line prompts the user to enter a list of numbers. The user is instructed to separate the numbers with spaces. The input is read as a single string and stored in the variable numbers\_input.
2. **numbers\_list = [float(num) for num in numbers\_input.split()]**: This is a list comprehension that converts the input string into a list of numbers.
   * numbers\_input.split() splits the input string into a list of substrings using spaces as the delimiter.
   * for num in ... iterates through each substring (which should be a number as a string).
   * float(num) converts each substring into a floating-point number.
   * [...] creates a new list called numbers\_list containing these floating-point numbers.
3. **mean\_value = sum(numbers\_list) / len(numbers\_list)**: This line calculates the mean (average) of the numbers in the numbers\_list.
   * sum(numbers\_list) calculates the sum of all numbers in the list.
   * len(numbers\_list) gets the number of elements in the list.
   * The sum is divided by the count to get the mean.
4. **maximum\_value = max(numbers\_list)**: This line uses the built-in Python function max() to find the largest number in the numbers\_list and stores it in maximum\_value.
5. **minimum\_value = min(numbers\_list)**: This line uses the built-in Python function min() to find the smallest number in the numbers\_list and stores it in minimum\_value.
6. **print("The list of numbers is:", numbers\_list)**: This line prints the list of numbers that was created from the user's input.
7. **print("The mean is:", mean\_value)**: This line prints the calculated mean value.
8. **print("The maximum value is:", maximum\_value)**: This line prints the calculated maximum value.
9. **print("The minimum value is:", minimum\_value)**: This line prints the calculated minimum value.

**TASK-2:**Write a python function that checks the given number is an Armstrong or not.

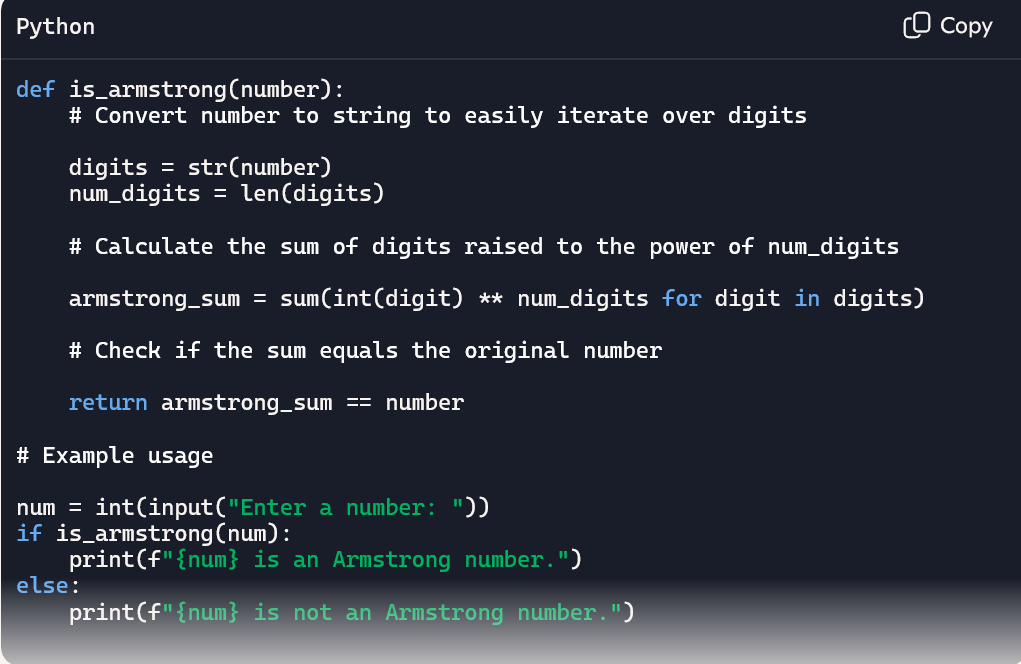
Comparision between gemini and copilot-

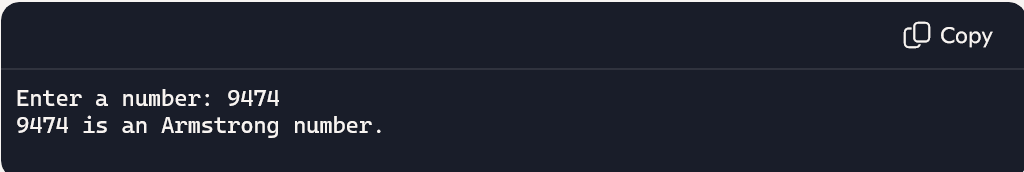
**GEMINI:**

****

****

**COPILOT:**

****

****

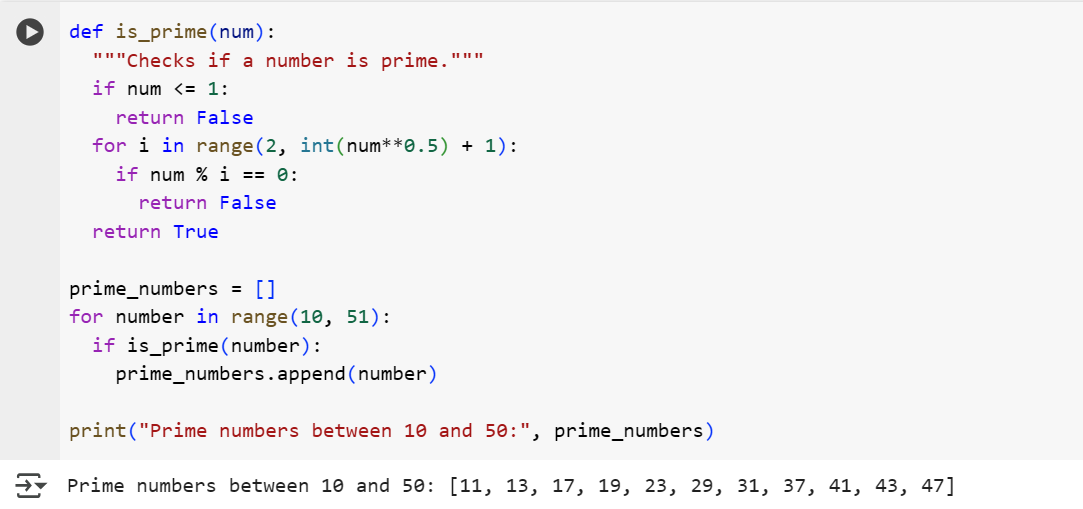
**EXPLANATION:**

1. **def is\_armstrong\_number(number):**: This defines the function that takes a number as input.
2. **num\_str = str(number)**: Converts the input number to a string so we can easily work with its digits.
3. **num\_digits = len(num\_str)**: Gets the total number of digits in the number.
4. **sum\_of\_powers = 0**: Initializes a variable to store the sum of the powers of the digits.
5. **for digit in num\_str:**: This loop goes through each character (digit) in the number string.
6. **sum\_of\_powers += int(digit) \*\* num\_digits**: Converts the current digit back to an integer, raises it to the power of the total number of digits (num\_digits), and adds it to sum\_of\_powers.
7. **return sum\_of\_powers == number**: After checking all digits, it compares the calculated sum\_of\_powers with the original number. If they are equal, the function returns True (it's an Armstrong number); otherwise, it returns False.

**TASK-3:**

Write a python program to generate prime numbers from 10 to 50 in the form of list.

**CODE AND OUPUT:**

****

**EXPLANATION:**

**1. The is\_prime function:**

* **def is\_prime(num):**: This defines a function named is\_prime that takes one argument, num.
* **"""Checks if a number is prime."""**: This is a docstring explaining the function's purpose.
* **if num <= 1:**: This checks if the input number num is less than or equal to 1. Prime numbers are defined as being greater than 1, so if the number is 1 or less, the function immediately returns False.
* **`for i in range(2, int(num**0.5) + 1):\*\*: This starts afor` loop to check for divisors.
  + range(2, int(num\*\*0.5) + 1) creates a sequence of numbers starting from 2 up to the integer part of the square root of num, plus 1. We only need to check for divisors up to the square root of a number because if a number num has a divisor greater than its square root, it must also have a divisor less than its square root.
* **if num % i == 0:**: Inside the loop, this checks if num is divisible by the current number i using the modulo operator (%). If the remainder is 0, it means i is a divisor of num.
* **return False**: If a divisor is found, the number is not prime, so the function immediately returns False.
* **return True**: If the loop completes without finding any divisors, it means the number is prime, and the function returns True.

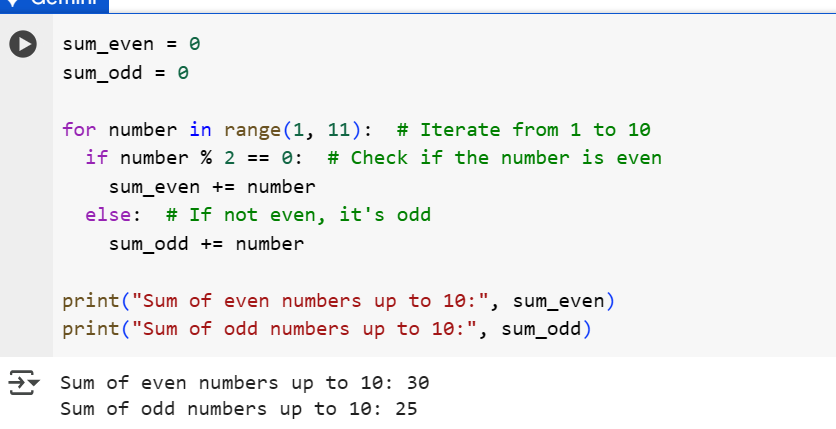
**2. Generating the list of prime numbers:**

* **prime\_numbers = []**: This line initializes an empty list called prime\_numbers. This list will store the prime numbers found within the specified range.
* **for number in range(10, 51):**: This for loop iterates through the numbers from 10 up to (but not including) 51. So it checks numbers 10, 11, 12, ..., 50.
* **if is\_prime(number):**: Inside the loop, for each number in the range, it calls the is\_prime function to check if that number is prime.
* **prime\_numbers.append(number)**: If the is\_prime function returns True (meaning the number is prime), that number is added to the prime\_numbers list using the append() method.
* **print("Prime numbers between 10 and 50:", prime\_numbers)**: After the loop finishes, this line prints a descriptive message followed by the prime\_numbers list containing all the prime numbers found between 10 and 50.

**TASK-4:**

**TASK-5:**Write a python program to get sum of even numbers upto 10 and sum of odd numbers upto 10

**CODE AND OUTPUT:**



**EXPLANATION:**

1. **sum\_even = 0**: This line initializes a variable called sum\_even to 0. This variable will store the running total of even numbers found.
2. **sum\_odd = 0**: This line initializes a variable called sum\_odd to 0. This variable will store the running total of odd numbers found.
3. **for number in range(1, 11):**: This is a for loop that iterates through a sequence of numbers generated by range(1, 11). The range(1, 11) function produces numbers starting from 1 and going up to (but not including) 11. So, the loop will process the numbers 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10. In each iteration, the current number is assigned to the variable number.
4. **if number % 2 == 0:**: Inside the loop, this if statement checks if the current number is even. The modulo operator (%) gives the remainder of a division. If a number divided by 2 has a remainder of 0, it means the number is even.
5. **sum\_even += number**: If the condition in the if statement is true (the number is even), this line adds the current number to the sum\_even variable. The += operator is a shorthand for sum\_even = sum\_even + number.
6. **else:**: This keyword indicates the block of code to execute if the condition in the if statement is false. In this case, it means the number is odd.
7. **sum\_odd += number**: If the number is odd, this line adds the current number to the sum\_odd variable. This is a shorthand for sum\_odd = sum\_odd + number.
8. **print("Sum of even numbers up to 10:", sum\_even)**: After the loop has finished iterating through all the numbers from 1 to 10, this line prints a descriptive message followed by the final calculated sum\_even.
9. **print("Sum of odd numbers up to 10:", sum\_odd)**: This line prints a descriptive message followed by the final calculated sum\_odd.